Wavelet Radiance Transport for Interactive Indirect Lighting

Janne Kontkanen, Emmanuel Turquin, Nicolas Holzschuch, François X. Sillion

Eurographics Symposium on Rendering (2006)
Tomas Akenine-Möller and Wolfgang Heidrich (Editors)

Wavelet Radiance Transport for Interactive Indirect Lighting

Janne Kontkanen[1], Emmanuel Turquin[2], Nicolas Holzschuch[2] and François X. Sillion[2]

[1]Helsinki University of Technology [2]ARTIS† GRAVIR/IMAG INRIA

Maze scene | Direct lighting | Indirect lighting | Complete illumination

Figure 1: *This scene is rendererd 15 FPS by our system with full global illumination. The light sources (spotlights) and the viewpoint can be modified interactively. The precomputation time was 23 minutes.*

Abstract

Global illumination is a complex all-frequency phenomenon including subtle effects caused by indirect lighting. Computing global illumination interactively for dynamic lighting conditions has many potential applications, notably in architecture, motion pictures and computer games. It remains a challenging issue, despite the considerable amount of research work devoted to finding efficient methods. This paper presents a novel method for fast computation of indirect lighting; combined with a separate calculation of direct lighting, we provide interactive global illumination for scenes with diffuse and glossy materials, and arbitrarily distributed point light sources. To achieve this goal, we introduce three new tools: a 4D wavelet basis for concise radiance expression, an efficient hierarchical pre-computation of the Global Transport Operator representing the entire propagation of radiance in the scene in a single operation, and a run-time projection of direct lighting on to our wavelet basis. The resulting technique allows unprecedented freedom in the interactive manipulation of lighting for static scenes.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Illumination simulation methods have many interesting applications, for example in architectural design, lighting design, computer games or motion pictures. These applications make use of global illumination algorithms, known for their high computational demands, and would greatly benefit from improved interactivity. We note that these applications are often dealing with indoor scenes, illuminated by local light sources whose position and orientation are subject to change. In this specific setup, interactive global illumination remains a particularly challenging issue.

However, by taking advantage of the linearity of the rendering equation it is possible to precompute light transport offline, and to use this data during run-time to obtain convincing global illumination effects: *Precomputed Radiance Transport* methods (PRT) [SKS02, Leh04] precompute the relationship between the emitted light and the radiance outgoing from the surfaces of the scene. In order to keep the complexity manageable, these methods usually express the emission in a low dimensional basis. The most common way to do this is to consider only infinitely distant lighting, and

† ARTIS is a research project in the GRAVIR/IMAG laboratory, a joint unit of CNRS, INPG, INRIA and UJF.

thus reduce the dimensionality of the emission to 2. Yet, local light sources (a.k.a. *near-field illumination*) have 5 degrees of freedom, that can be narrowed down to 4 without loss of generality if we consider that light travels through a vacuum; this high dimensionality tends to make classical PRT methods extremely costly.

In this paper, we present a technique for interactive computation of global illumination in static scenes with diffuse and glossy materials, and arbitrarily placed dynamic point/spotlights. Our algorithm uses a precomputed *Global Transport Operator* that expresses the relationship between incident and outgoing surface radiance. During run-time we project the direct light form the light sources to the surfaces, and apply this precomputed operator to get full global illumination. Rather than following the common *compute, then compress* scheme, we try to generate the operator directly in a compact representation.

Our contributions are: a new 4D wavelet basis for compact representation of radiance, a method to efficiently compute the Global Transport Operator, greatly speeding up the precomputation time, and a method to efficiently project direct lighting from point light sources on our hierarchical basis at runtime. These three contributions, combined together, result in interactive manipulation of light sources, with immediately visible results in the global lighting.

The most noticeable limitation of our approach is directly linked to a well-known problem of finite-element methods for global illumination: our basis functions have to be expressed on the surfaces of the scene. Incidentally, our example scenes are exclusively composed of large quads. Another important limitation is that BRDFs must be relatively low-frequency to be efficiently representable in our wavelet basis.

2. Previous work

Global illumination has been the subject of research in Computer Graphics for decades. Dutré *et al.* [DBB03] give a complete survey of the state-of-the art of global illumination techniques. There have been plentiful research efforts to speed up global illumination computations and achieve real-time or interactive framerates.

Ray-tracing has been ported to the GPU [PBMH02, PDC*03] or to specific architectures [WSB01, SWS05]. The same has been done with the radiosity algorithm [Kel97, CHL04], while others use the GPU for fast computation of hierarchical form-factors [LMM05].

Nijasure *et al.* [NPG05] compute a representation of the incident radiance at several sample points sparsely covering the volume enclosed by the scene. Incident radiance is stored using spherical harmonics. Spherical harmonics coefficients are interpolated between the sample points and applied to the surfaces of the scene. The system can be iterated to compute multiple bounces of light, at the expense of rendering time. This approach can be seen as a dynamic generalization of Greger's *irradiance volumes* [GSHG98].

Dachsbacher and Stamminger [DS06] introduce an extended shadow map to create first-bounce indirect light sources. They splat the contribution of these sampled sources onto the final image using deferred shading. They only compute the first indirect light bounce, without taking visibility into account. Nevertheless, they observe that the results look plausible in most situations.

Despite using GPUs or even custom hardware, the above methods currently barely run interactively, unless they restrict themselves to small scenes or degrade the accuracy of the simulation.

In a separate research direction, PRT techniques [Leh04] precompute light exchanges and store the relationship between incoming lighting and outgoing global illumination on the surface of an object. The result of these precomputations, the light transport operator, is compressed and used at runtime for interactive display of global illumination. Most PRT techniques start by precomputing the light transport operator with great accuracy, *then* compress it, typically using *clustered principal component analysis* [SHHS03].

The cost of the uncompressed light transport is directly related to the degrees of freedom (DOF) n in the operator, growing with $O(k^n)$. As discussed in section 1, the general expression for emission space, assuming no participating media, has 4 DOF. Given that the outgoing surface radiance also has 4, the general form of the operator end up with 8 DOF.

To keep memory and precomputation costs tractable, most PRT techniques somehow restrict these degrees of freedom. It is generally achieved by assuming infinitely distant lighting, as done Sloan *et al.* [SKS02] and many others. Another option is to fix the locations of the light sources in space [DKNY95]. Yet another one is to fix the viewpoint [NRH03]: in this work, Ng *et al.* demonstrate that all-frequency lighting from an infinitely distant environment can be rendered efficiently by using a light transport operator expressed in Haar wavelet basis and non-linearly compressed. The fixed viewpoint restriction applies when the scene contains glossy materials. In a subsequent publication [NRH04] the authors remove this restriction by introducing triple wavelet product integrals. As a result they are able to generate high quality pictures that solve the 6-dimensional transport problem, but not with real-time or interactive rates.

Haar wavelets have succesfully been used by others [LSSS04, WTL06] to efficiently express all-frequency transport from detailed environment maps to glossy surfaces. These method utilize separable decomposition, consisting of a purely light-dependent term and a purely view-dependent term.

The approaches closest to ours are those of Kristensen *et al.* [KAMJ05] and Hasan *et al.* [HPB06]. Both consider static scenes under near-field illumination, and they separate the computation of direct lighting, done on the fly using standard GPU-based methods, and indirect lighting precomputed on some specific basis functions. Kristensen *et al.* [KAMJ05] use a 3D unstructured point cloud basis, precomputing radiance transport from this basis to the surfaces of the scene. At run-time, uniform point light sources are projected onto the point-cloud basis, then they apply the precomputed transport operator to obtain indirect illumination on the surfaces.

In a work concurrent to ours, Hasan *et al.* [HPB06] precompute direct-to-indirect transport corresponding to our GTO and express it in wavelet basis. The receiving basis consist in the visible pixels, and the sending basis is build by distributing point samples into the scene, which are then hierarchically clustered. a preprocess. Each of these methods presents different limitations: the former is restricted to omni-directional point lights, and the later renders high-quality pictures but only for diffuse-to-diffuse indirect transfer (although the last reflection can be arbitrarily glossy) and fixed viewpoint. As a comparison, our method doesn't suffer from these restrictions, but is limited to simple geometry and works best with diffuse materials.

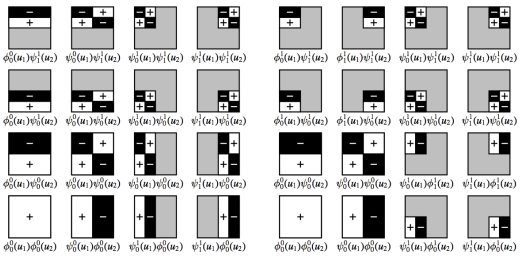
A common problem to many of the existing PRT methods is their fairly inefficient approach to precomputation. A lot of information computed during this step is discarded during a subsequent compression stage. One of our main concerns is precisely to avoid unnecessary computations and rather try to directly generate a concise operator (Hasan *et al.* share this objective). This way, we greatly reduce the memory cost and computation time for the precomputation step. We show clear improvement in precomputation times compared to Kristensen *et al.*, but as stated earlier, our finite element approach also brings restrictions not present in their work.

Our work draws inspiration from hierarchical finite element methods for global illumination [HSA91, GSCH93]. Wavelet and hierarchical algorithms adapt the solution to the geometry and lighting conditions: a coarse resolution is used where the illumination is smooth, and a finer resolution when there are sharp variations.

In our precomputation, we adopt some of the solutions used in Wavelet Radiance [CSSD94, CSSD96] which solves global illumination using a hierarchical 4D wavelet basis. Wavelet Radiance uses *non-standard* decomposition to represent surface radiance, but a *standard* decomposition for the transport operator. The latter enables direct computation of the transfer coefficients without needing a push-pull step.

3. Overview of the algorithm

Our method takes as input the geometric definition of a static and easily parametrized scene (*e.g.* composed by quads), and



(a) Standard refinement (b) Non-standard refinement

Figure 2: *Forming multi-dimensional wavelet basis.*

provides interactive visualization of global illumination effects in this scene under dynamic local lighting. We compute a 4D wavelet representation of indirect radiance on the surfaces of the scene. Our technique can be split into two high-level components: an offline component for precomputing the light transport operator, and a run-time component for rendering indirect lighting using this operator.

The run-time component uses the precomputed transport operator to interactively render global illumination:

1. Project direct lighting onto our wavelet basis.
2. Apply the precomputed global transport operator, resulting in indirect lighting, expressed in our wavelet basis.
3. Convert this result to outgoing radiance, and blend with direct lighting.

We treat direct lighting separately because it contains sharp details that are best rendered using specific techniques. Our run-time component is explained in section 5.

The offline component consists of these two steps:

1. Compute a *Direct Transport Operator* (DTO) for the scene. The DTO expresses the propagation of light inside the scene, and corresponds to a single bounce of light.
2. Compute a *Global Transport Operator* (GTO) using the DTO. The GTO expresses the full radiance propagation inside the scene, *i.e.* an infinite number of light bounces.

For efficient computation and compact representation, we express the operators in wavelet basis. Our bases for expressing the surface radiance and the operators are described in section 4.2. The computation of the DTO is explained in detail in section 4.3. In section 4.4 we discuss how the DTO can be used to efficiently compute the GTO using Neumann series and non-linear compression.

4. Offline component

4.1. Wavelet Basis for Surface Radiance

Surface radiance is a 4-dimensional function: two dimensions for the surface location and two dimensions for the direction. Because of its 4D nature, storing a tabulated version of surface radiance is prohibitively costly. This problem is

even more pronounced for the 8-dimensional transport operator that expresses the radiance transport between surfaces.

An efficient expression and computation of the transport operator highly benefits from a hierarchical representation; hence we chose wavelets as our basis functions. We elected Haar wavelets for their computational simplicity, but our algorithm can be applied to any type of *tree* wavelet basis [GSCH93].

The building blocks of Haar basis are the following smooth function ϕ and wavelet ψ:

$$\phi(x) = \begin{cases} 1 & \text{for} \quad 0 \le x < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\psi(x) = \begin{cases} 1 & \text{for} \quad 0 \le x < 1/2 \\ -1 & \text{for} \quad 1/2 \le x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

All the wavelets and smooth functions of Haar basis are formed by scales and translates of the above elementary functions as follows:

$$\phi^i_j = \phi(2^i x - j), \psi^i_j = \psi(2^i x - j) \tag{2}$$

Where i gives the scale, and j gives the translation. For comprehensive introduction to Haar wavelets and wavelets in general we refer to [SDS96].

Multi-dimensional wavelet bases are usually formed by combining one-dimension wavelet bases. There are two systems for creating multi-dimensional wavelet bases: the standard refinement (see Figure 2a), where the dimensions are refined separately, and the non-standard refinement (see Figure 2b), where refinement is performed alternatively along all dimensions.

The non-standard refinement method merges together the different dimensions, treating them equally. As a consequence, it is more widely used in fields such as Image Analysis and Image Synthesis, where the two spatial dimensions serve an equal purpose.

For Radiance computations, the spatial and angular dimensions are *not* equivalent. A surface can exhibit large variations on the spatial domain and be more continuous over the angular domain, and linking the resolutions of the spatial and angular dimensions is not always efficient. For this reason, we decouple the spatial and angular domains, using standard refinement between these dimensions. For the 2D sub-domains for angular and spatial dimensions, we still use non-standard refinement. Our wavelet basis for 4D radiance therefore uses a combination of standard and non-standard refinement.

For the angular domain, the hemisphere of directions is mapped to the unit square using a cosine-weighted concentric map [SC97]; we then apply wavelet analysis over the unit square. Using this mapping allows pre-integrated cosine on the hemisphere of directions, with a low angular distortion and constant area mapping.

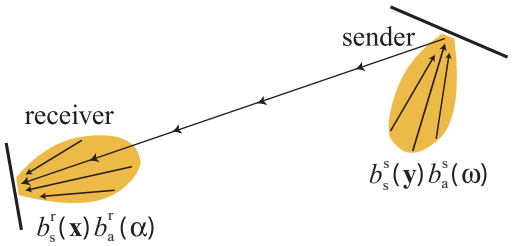


Figure 3: *Light transport from sending basis function $b^s_s(\mathbf{y})b^s_a(\omega)$ to receiving basis function $b^r_s(\mathbf{x})b^r_a(\alpha)$.*

4.2. Wavelet Basis for Transport Operator

The projected transport operator consists of coefficients that describe the influence of each basis function to all the other ones. The non-standard operator decomposition is a more common choice in hierarchical radiosity, as in theory it gives a more compact representation than standard decomposition. In spite of this we chose to follow [CSSD94] and used standard operator decomposition. We see two advantages in using the standard decomposition: it decouples the resolution for sender and receiver, and there is no need for a push-pull step. The former is an obvious advantage, for example when the sender and receiver differ greatly in size or in complexity.

The latter requires an explanation: conventional global illumination methods, using the non-standard representation, require a push-pull step between light bounces [HSA91]. For these methods, the cost of the push-pull step is not prohibitive. However, we are using the DTO to compute the Global Transport Operator, using Neumann series (see section 4.4). During this computation, we perform several multiplications between operators. During these operator multiplications, the fact that we do not require a push-pull step greatly accelerates the computation.

4.3. Direct Transport Operator

We compute the Direct Transport Operator to express a single bounce of light. As we are going to conduct operator multiplications, we require the output space of the DTO to be equal to its input space. This leaves a choice: either we express the DTO in terms of incident radiance or in terms of outgoing radiance. We chose to use the incident form of the Direct Transport Operator.

The incident form of the Direct Transport Operator is defined as follows:

$$(\mathcal{T}L)(\boldsymbol{x}, \boldsymbol{x} \leftarrow \boldsymbol{y}) = \int f_r(\omega, \boldsymbol{y}, \boldsymbol{y} \rightarrow \boldsymbol{x}) V(\boldsymbol{x}, \boldsymbol{y}) \lfloor \omega \cdot \boldsymbol{n}_y \rfloor L(\boldsymbol{y}, \omega)\, \mathrm{d}\omega \tag{3}$$

The transport operator maps the incident radiance arriving to location $\boldsymbol{y}$ from direction ω to incident radiance at another location $\boldsymbol{x}$ from direction $\boldsymbol{x} \leftarrow \boldsymbol{y}$. Given a certain distribution of incident radiance, applying this operator once gives the distribution of light that has been reflected once from the

surfaces of the scene. Here f_r refers to BRDF and V to the visibility term. Along with the $\lfloor \omega \cdot \boldsymbol{n_y} \rfloor$ term, they form the kernel $k(\boldsymbol{x}, \boldsymbol{y}, \omega)$ of the light transport operator.

The projected form of the transport operator is obtained by integrating the 6D kernel against each 8D wavelet, in a similar fashion to [CSSD94]:

$$\int K(\boldsymbol{x}, \boldsymbol{y}, \omega) b_s^s(\boldsymbol{y}) b_a^s(\omega) b_s^r(\boldsymbol{x}) b_a^r(\boldsymbol{x} \leftarrow \boldsymbol{y}) \, \mathrm{d}\omega \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{y} \qquad (4)$$

Where $K(\boldsymbol{x}, \boldsymbol{y}, \omega) = k(\boldsymbol{x}, \boldsymbol{y}, \omega) \frac{\lfloor \boldsymbol{x} \leftarrow \boldsymbol{y} \cdot \boldsymbol{n_y} \rfloor}{r_{xy}^2}$ and b_s^r, b_a^r, b_s^s and b_a^s refer to the elementary non-standard basis functions of receiving spatial, receiving angular, sending spatial and sending angular dimensions, respectively (see section 4.1). $\boldsymbol{x}$ and $\boldsymbol{y}$ are integrated over surfaces, while ω is integrated over the hemisphere oriented according to corresponding surface normal. For a visual illustration, see Figure 3.

In the context of light transport, a wavelet coefficient obtained from Equation 4 has traditionally been called a *link*. We will use this term to refer to a group of coefficients for 8D basis functions sharing the same support on all 2D subspaces. In practice, this means that each link corresponds to 255 wavelets and a single smooth function coefficient. This can be seen by considering that each 2D sub-space has 4 elementary non-standard basis functions that share the same support, and $4^4 = 256$. As an example of elementary 2D functions, see the four functions in the lower left corner of Figure 2b.

We compute the Direct Transport Operator by progressively refining the existing links. We start by creating interactions between coarsest level basis functions in the scene, and then refine these. At each step, we consider 256 basis function coefficients. Note, however, that not all the 256 coefficients are stored. We only store the necessary parts of link: a link between two diffuse surfaces does not need wavelets in angular domain. In practice, each link contains between 1 and 256 wavelet coeffients depending on its type.

A refinement oracle (see section 4.3.2) tells us whether a link needs to be refined. For each link it has a choice to refine in any of the 2D sub-domains (spatial receiver, angular receiver, spatial sender, angular sender). The refinement oracle may independently choose each option, possibly refining both the sender and the receiver in space and angle, or simply refining the receiver in space, or any combination.

When the link is refined, we create the child wavelets, and recursively consider each newly created link. Consider a refinement of one of the spatial basis functions: when a spatial basis function is refined, four new child links are created (spatial patch is divided into four child patches). However, when two of the 2D sub-domains are refined, there will be $4 \times 4 = 16$ child links to consider, and finally if all the dimensions are refined $4^4 = 256$ child links are created.

When performing progressive refinement in standard basis, the refinement may arrive at a certain link from several parent links. This means that when we chose the standard method for combining dimensions in our 8D basis, we partially lost the tree-property of our basis.

However, we use the same solution for the problem as was used in the conventional wavelet radiance [CSSD94]. If, in the refinement process, we arrive at a 8D wavelet coefficient that has already been visited, we terminate the traversal. The difference with conventional wavelet radiance is that we have four independent subspaces instead of two.

An important point in our algorithm is that, as with Wavelet Radiance [CSSD94], we are computing wavelet transport coefficients directly between wavelet coefficients, not between smooth functions. This eliminates the need for push-pull step.

4.3.1. Numerical Integration

The actual coefficients corresponding to each link are computed by generating quasi-randomly distributed samples in the support area of the link. Thus, we are computing Equation 4 by quasi-Monte Carlo integration.

The coefficients of the coarsest links are difficult to compute accurately without a significantly large amount of samples. On the other hand the finer scale wavelets do not require as many samples since within a smaller support the kernel does not deviate as much. Because of this we adopt the adaptive integration procedure used in Wavelet Radiance [CSSD94]: we first refine the link structure to the finest level and then perform a wavelet transform to compute the coarser links in terms of the finer ones. As a result, only the finest scale wavelet coefficients are computed directly. In our implementation, this procedure is done during a single recursive visit.

4.3.2. Refinement Oracle

The refinement oracle considers each link, *i.e.* a cluster of coefficients of wavelets sharing the same support at the time. It works by testing quasi-random samples of the kernel, and using explicit knowledge of the BRDF. If the oracle finds that the operator is smooth, then the refinement stops and the kernel samples are used to compute the wavelet coefficients.

At each refinement step, the refinement oracle has to select whether to refine the sender or the receiver, or both, and whether to refine them spatially or angularly, or both. Thus, the oracle can refine between 0 to 4 dimensions, resulting in 16 possible combinations. The ability to make an independent refinement decision in each sub-space is a consequence of using standard refinement as described in sections 4.1 and 4.2.

The decision to refine the interaction in the angular domain is based solely on the BRDFs of the sender and receiver, unless the sender and the receiver are mutually invisible, in which case the interaction is not refined. The basis

functions are mutually invisible if no unobstructed ray can be generated between the supports of the sender and the receiver. Note that this can happen even if the spatial basis functions are not mutually occluded, but the angular support of the sending basis function is oriented in such a way that it does not point towards the receiving basis function.

Diffuse surfaces are never subdivided angularly. For an arbitrarily glossy BRDF, the maximum level of angular subdivision depends on the resolution of its wavelet representation. Note that at this point of the algorithm, the wavelet representation is only used to control the refinement resolution, whereas the kernel samples are evaluated using the original, possibly analytic representation of the BRDF.

Spatial refinement is based on the kernel deviation estimated on the samples. To take advantage of the standard operator decomposition, *i.e.* the ability to refine the sender and receiver independently, we apply the following heuristics:

- Refine sender if $(\max(K) - \min(K))A_s > \varepsilon_s$
- Refine receiver if $(\max(K) - \min(K))A_r > \varepsilon_r$

With K as defined in Equation 4; A_s, A_r refer to the surface areas covered by the supports of the basis functions; and ε_s, ε_r to the user selected thresholds. We use separate thresholds for sending and receiving refinement, since it is useful to generate asymmetrically refined matrices, where the sending basis functions are coarser than the receiving (see section 4.4).

4.4. Global Transport Operator

Having computed the direct transport operator $\mathcal{T}$, which expresses a single bounce of light transfer between the surfaces in the scene, we use it to compute the global transport operator, using the Neumann series:

$$\mathcal{G} = \mathcal{I} + \mathcal{T} + \mathcal{T}^2 + \mathcal{T}^3 + \mathcal{T}^4 + \ldots$$

The global transport operator expresses the relationship between the converged incident lighting and the incoming incident lighting. $\mathcal{G}$ is computed iteratively, from $\mathcal{T}$. At each step, we compute $\mathcal{T}^{n+1} = \mathcal{T}^n\mathcal{T}$.

The computation of above series is rather expensive using a high resolution representation of $\mathcal{T}$, since in the end all the basis functions interact with each other (unless the scene consists of separate local environments with no mutual visibility to each other). For this reason, we aggressively compress the matrices during the computation: after each computation of $\mathcal{T}^n$, we apply non-linear compression to the result, removing all coefficients below a certain threshold.

Because of the compression, the number of coefficients in $\mathcal{T}^n$ decreases when n increases (see Figure 6). We stop the computation when all the coefficients in $\mathcal{T}^n$ are smaller than our threshold, which in our experiments required up to ten iterations. To speed-up the computations, we pre-multiply by the sparsest matrix, computing $\mathcal{T}^n$ times $\mathcal{T}$.

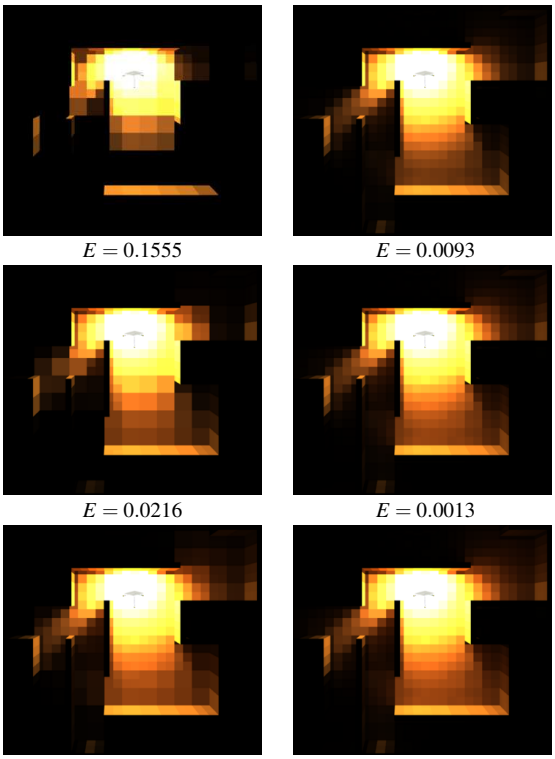


Figure 4: *Multiplying the GTO by the original fine-resolution DTO (right) improves the visual quality compared to the original GTO (left). Notice that even with a larger numerical error, the gathered GTO gives a more pleasing result (compare upper right corner with lower left corner). Error E refers to the sum of squared differences of wavelet coefficients when compared to uncompressed GTO.*

After a coarse GTO has been computed, we still perform one more step to improve the results: we multiply the series by the original fine-resolution DTO from the left. This operation can be thought as a kind of final gathering that improves the visual aspect of the result by using higher resolution representation for the last bounce before the light meets the eye (see Figure 4), in the same spirit as Hasan *et al.* [HPB06].

5. Run-time component

The run-time component of our method works as follows:

1. Project direct lighting on the wavelet basis defined on the surfaces of the scene (section 5.1).
2. Use the precomputed GTO to transform the projected direct light to the converged incident radiance (section 5.2).
3. Transform incident radiance into outgoing radiance by applying the BRDF of the surfaces (section 5.3).
4. Render the indirect light using the wavelet basis and combine it with direct light computed separately (section 5.4).

5.1. Direct Light Projection

In order to use the GTO to generate indirect lighting, we need to project the light from the dynamic light sources to the 4D radiance basis defined on the surfaces of the scene.

For each light source and for each surface of the scene, our method proceeds as follows:

1. Estimate the level of precision required.
2. Compute all the smooth coefficients at this level of precision, by integrating direct lighting on the support of each coefficient.
3. Perform a wavelet transform on these smooth coefficients to compute the wavelet coefficients, then discard the smooth coefficients. This generates a wavelet representation of the direct light on all the surfaces of the scene.

This projection of direct lighting onto our wavelet basis is fundamental for interactive rendering, so it is important to perform these computations efficiently. Unfortunately, step 2 involves computing direct lighting for all the smooth coefficients, a costly step for arbitrary light sources.

To estimate the level of precision required (step 1), we look at the solid angle subtended by the geometry of the object, multiplied by the intensity of the light source in the direction of the object.

The computation of the smooth coefficients (step 2) involves computing direct lighting in the scene, including visibility between the light source and the support of the smooth coefficient. In our implementation, we tried both area light sources and point light sources, but we found that only point light sources were currently compatible with interactive framerates.

For point light sources, we compute the visibility using occlusion queries, *i.e.* we render the smooth functions from the view of light source using *GL_ARB_occlusion_query* extension of OpenGL, and estimate the solid angle each basis function subtends based on the number of visible pixels. Our current implementation only supports direct light projection to directionally smooth basis functions. This means that direct light falling on a specular surface gets reflected as if the surface was diffuse.

To benefit from our sparse wavelet representation for surface radiance, the elements need to be dynamically (de-)allocated. To avoid an excessive amount of dynamic memory management we use the following method: before projecting the direct light at each frame we set the existing allocated coefficients to zero. Then we project the light as described, and after the projection we de-allocate the entries that are still null. This minimizes the amount of dynamic allocations and de-allocations required during run-time.

5.2. Application of the GTO

Once we have a wavelet projected representation of direct incident lighting, we multiply it by the GTO to give the converged incident radiance:

$$\boldsymbol{X} = \mathcal{G}\boldsymbol{E}$$

where $\boldsymbol{E}$ represents the projected direct light, $\mathcal{G}$ is the GTO, and $\boldsymbol{X}$ is the resulting converged incident radiance. All the wavelet representations above are in sparse format, so that only non-zero coefficients are stored.

For efficient multiplication, it is important to take advantage of the sparseness of $\boldsymbol{E}$: typically the direct light can be expressed with a small number of wavelet coefficients, since it is often either spatially localized or falling from a far away light source, in which case only coarse basis functions are present in $\boldsymbol{E}$. We perform the multiplication by considering only the non-zero element of E and accumulating the results to $\boldsymbol{X}$.

We use the same technique to minimize the amount of dynamic memory allocations in $\boldsymbol{X}$ as we used for computing $\boldsymbol{E}$ (section 5.1).

5.3. Multiplication by the BRDF

$\boldsymbol{X}$ represents the incident indirect radiance, and yet we need the outgoing radiance for display. Thus, we need a final multiplication by the BRDF. In our implementation, we associate a wavelet representation of the BRDF with each surface of the scene and this step simply translates into a multiplication in wavelet space. Note that we use the same wavelet representation that the oracle uses to determine the angular refinement (section 4.3.2).

5.4. Rendering from the Wavelet Basis

To generate the final view for the user, we first render the scene representing the indirect light, and then additively blend in the direct light using standard techniques.

The indirect light is synthesized from the 4D wavelet basis to textures using the CPU. Then the whole scene is drawn using standard texture mapping and optionally bi-linear filtering (results without this filtering can be seen in Figure 4).

To get rid of the discontinuities that would appear between neighboring coarse level quads, we use border texels (supported by standard graphics hardware) to ensure a smooth reconstructed result across the edges of quads.

Each quad is associated with its own texture, and thus it is possible to use a specific texture resolution for each quad. In our current implementation we select the texture resolution according to the maximum of the spatial and angular resolutions present in the wavelet basis.

The texture synthesis is performed by traversing all non-zero wavelet coefficients for a given quad. For performance,

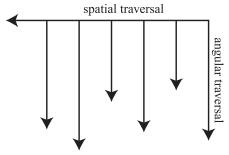


Figure 5: *For texture synthesis, we traverse the wavelet hierarchy in the order shown here. We terminate the angular traversal as soon as we detect that the angular sub-tree points away from the viewer.*

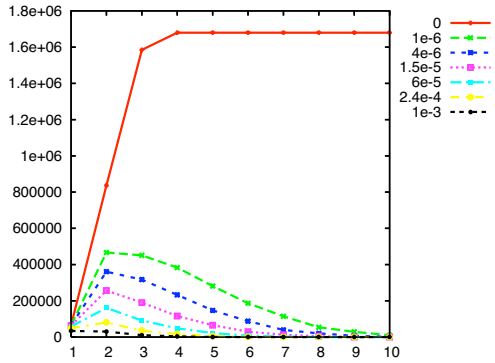


Figure 6: *Number of non-zero entries in DTO^n, as a function of number of bounces n, depending on the threshold used for wavelet compression, for the maze scene.*

we exploit our knowledge of the view direction and avoid traversing the subtrees of wavelets that do not point towards the eye: we traverse the wavelet hierarchy first in spatial order, then in angular order (see Figure 5), and terminate the angular traversal if we detect that the whole subtree points away from the viewer.

6. Experiments and comparison

We computed the GTO at different resolutions for three different scenes: the Cornell box, a maze and a simple scene for testing glossy illumination (see Figure 10). The results are summarized in Table 1. As can be seen, all the results run either in real-time or at least at interactive framerates.

6.1. Offline component

The most important result is the speed of the precomputation step. For comparison, the maze scene we used is an exact replica of the scene used by Kristensen *et al.* [KAMJ05]. The time it takes for our method to compute the GTO on this scene varies between 24 and 74 minutes depending on the required quality. Kristensen *et al.* report a precomputation time of 6.5 hours on a cluster of 32 PCs. Since their method is easily parallelizable, we may assume that the performance is almost linear with the number of machines, translating into a total computation time of approximately 8 days using a single PC. The comparison is based on a visual judgement. For more exact evaluation, a numerical error metric would need

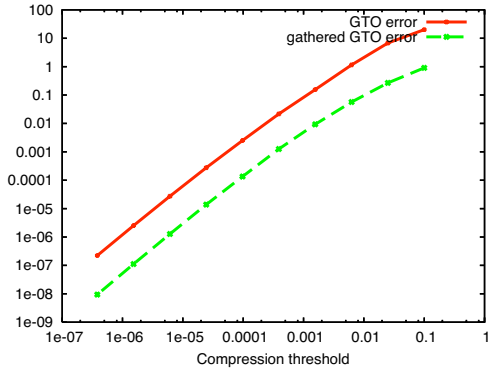


Figure 7: *GTO error in the maze scene as a function of the threshold on wavelet coefficients.*

to be used. Nevertheless, we believe it is fair to say that our method performs the precomputation faster.

This acceleration comes from our algorithm's ability to avoid the computation of unnecessary data. All the information computed during the pre-processing step is used for the runtime computation of indirect lighting. On the other hand, our technique suffers from well-known issues in finite element methods: we need a parameterization of our scene, which restricted us to easily parametrizable surfaces (quads in our current implementation).

Our precomputation time is dominated by the hierarchical refinement to compute the DTO, while the Neumann series evaluation to compute the GTO is relatively fast. The threshold used for non-linear wavelet compression in the GTO computations has an immediate impact on the memory cost of our algorithm (see Figure 6): not using compression in the maze scene results in approximately 2 million links stored. As each link stores 9 to 16 wavelet coefficients in floating point and in three channels, the average cost of a link is 150 bytes. Thus these 2 million links correspond to approximately 300 Mb which is not practical for real-time use.

Even a very small threshold ($\varepsilon = 10^{-6}$) on our wavelet coefficients brings the number of links down to 400,000, corresponding to memory cost of 60 Mb. A more aggressive compression ($\varepsilon = 10^{-4}$) further divides these numbers by 6, bringing the memory cost to 10 Mb.

We checked the relationship between the level of wavelet compression used on the GTO and the error we make on the operator. We tested both the standard GTO and the "gathered GTO", where the GTO is premultiplied by the original, fine-resolution DTO. Using the non-compressed operator as a reference, we computed the error as a function of the threshold used for compression (see Figure 7). The error on both operators decreases regularly with the threshold, with the error on the gathered GTO being consistently smaller than the error on the standard GTO.

We also checked the relationship between the memory

Table 1: *Summary of the performance of our algorithm. All matrices were computed with a single 3 GHz Pentium 4.*

	CORNELL	CORNELL HI-RES	MAZE	MAZE HI-RES	GLOSSY	GLOSSY HI-RES
t_{DTO} (precomp.)	2min	25min	23min	1h 12min	1min	9min
t_{GTO} (precomp.)	<1s	2s	40s	1min	<1s	1min
FPS (run-time)	60	25	15	7	8	3
Links DTO	3477	30366	65501	288628	4260	36778
Links GTO	418	648	24151	24599	1712	34176
Links gathered GTO	14169	53100	164813	589361	44383	195037
Memory cons. in MB	1.7	6.4	19.7	70	5.3	23.4

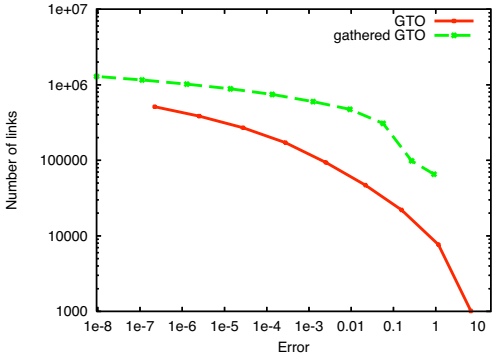


Figure 8: *Memory cost of the GTO as a function of the error on the operator (maze scene).*

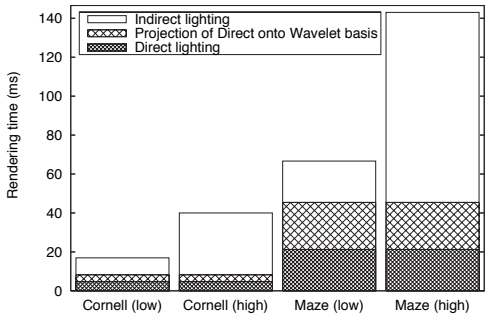


Figure 9: *Rendering times for the different steps of our runtime component. For each scene, we tried a high resolution (moderately compressed) and a low resolution (aggressively compressed) GTO.*

cost of the GTO and the error it represents (see Figure 8). For both versions of the GTO, the error decreases as the memory cost increases. We observe that, surprisingly, the GTO outperforms the gathered GTO: for a given error, it always provides a more compact representation of the operator. Even so, this compact representation does not always translate into visual quality (see Figure 4): comparing the two representations of the GTO with similar error levels, we found that the gathered GTO gives better visual results. The non-linear compression we used for computing the GTO removes links based on the energy they represent. However, the visual quality of the image is not directly linked to energy levels, but also to other spatial information.

6.2. Runtime component

We analyzed the performance of our runtime component. Timings for each step can be seen on Figure 9. We tried two different scenes, each of them with a different level of compression of the GTO. All these rendering times correspond to observed framerates.

Computations related to direct lighting only dependent on scene complexity, as the projection of direct lighting does not depend on the accuracy on the GTO. Computing the projection of direct lighting is about as expensive as the computation of visible direct lighting on the GPU.

Not surprisingly, the computations related to indirect lighting (multiplying the direct lighting by the GTO, the multiplication by the BRDF and conversion to textures) dominate, especially when using a high quality GTO (moderate compression). We observe that the rendering time for indirect lighting is related to the number of coefficients in the GTO.

7. Conclusions and Future work

In this paper, we have presented a novel algorithm for fast computation of indirect lighting. Combined with a separate computation of direct lighting, our algorithm allows interactive global illumination.

Our algorithm makes use of three different contributions: first a new wavelet basis for efficient storage of radiance, using standard refinement to separate the angular and spatial dimensions, secondly a hierarchical precomputation method for PRT, and third a fast projection of direct lighting on our basis. Our method works in a top-down approach, and therefore aims to only compute the information that is necessary for PRT computations.

Our main limitation is inherited from the finite element methods: the elements (*i.e.* the basis functions) need to be mapped to the surfaces of the scene. In the simplest case, the scene needs to initially consist of large quads as in the examples of this paper. Nonetheless, as a future work we wish to study the possibility of to relieve these restrictions in the spirit of bi-scale radiance transfer [SLSS03]. This would require an easily parameterizable coarse version of the scene

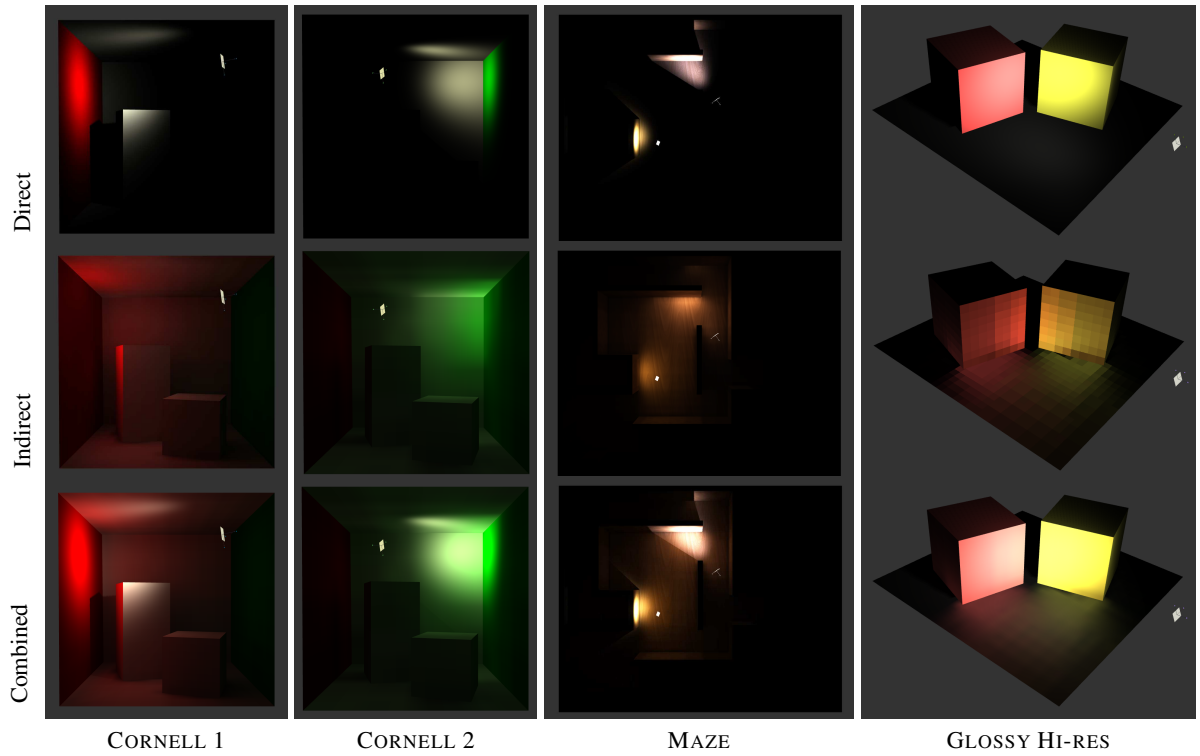


Figure 10: *Example results on our test scenes. In the glossy case (right column), we show the indirect illumination non-filtered, and the composited image using bi-linear filtering.*

and a method to transfer lighting from this coarse surface to a finer one.

Another direction for future work is establishing more explicit relationship between the different compressions done in our algorithm. In the current method, we have separate thresholds for hierarchical refinement and the non-linear compression used during the Neumann series computation. It would be advantageous to only have a single threshold related to the quality of GTO we want to obtain.

The oracle's ability to independently refine each subspace is both a strength and a challenge. The refinement heuristics we presented are not optimal, since they do not take into account the dependence of directional and spatial dimensions, as explained by Durand *et al.* [DHS*05]. For instance, it might not make sense to link a sender with a narrow angular support to a spatially large receiver. We believe that this is a very promising direction for future research and that clear performance improvements are possible.

Yet another idea concerns applying knowledge of runtime importance, projected from the camera to the surfaces of the scene. This projection could be used to speed-up the GTO multiplication as we would know beforehand which basis functions really contribute to the image. So we could use only the parts of the GTO that actually have an visible effect on the result.

Finally, to leverage the full potential of our 4D representation, we plan to explore run-time projection of arbitrarily distributed area light-sources instead of point lights. This would have to be coupled with an accurate display of direct lighting, which could benefit from the information we collect during the projection.

Acknowledgements

We would like to thank the Computer Graphics Group of Helsinki University of Technology, the ARTIS team, and our anonymous reviewers for their valuable feedback. This work has been supported by Bitboys, Hybrid Graphics, Remedy Entertainment, Anima Vitae, and the National Technology Agency of Finland.

References

[CHL04] COOMBE G., HARRIS M. J., LASTRA A.: Radiosity on graphics hardware. In *Graphics Interface 2004* (2004).

[CSSD94] CHRISTENSEN P. H., STOLLNITZ E. J., SALESIN D. H., DEROSE T. D.: Wavelet Radiance. In *Photorealistic Rendering Techniques (Proc. of EG Workshop on Rendering)* (June 1994), pp. 287–302.

[CSSD96] CHRISTENSEN P. H., STOLLNITZ E. J.,

SALESIN D. H., DEROSE T. D.: Global Illumination of Glossy Environments Using Wavelets and Importance. *ACM Transactions on Graphics 15*, 1 (Jan. 1996), 37–71.

[DBB03] DUTRÉ P., BALA K., BEKAERT P.: *Advanced Global Illumination*. AK Peters, 2003.

[DHS*05] DURAND F., HOLZSCHUCH N., SOLER C., CHAN E., SILLION F. X.: A frequency analysis of light transport. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2005) 24*, 3 (Aug. 2005).

[DKNY95] DOBASHI Y., KANEDA K., NAKATANI H., YAMASHITA H.: A quick rendering method using basis functions for interactive lighting design. *Computer Graphics Forum (Proc. of EUROGRAPHICS 1995) 14*, 3 (Sept. 1995).

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *Interactive 3D Graphics 2006* (2006).

[GSCH93] GORTLER S. J., SCHRODER P., COHEN M. F., HANRAHAN P.: Wavelet Radiosity. In *SIGGRAPH '93* (1993), pp. 221–230.

[GSHG98] GREGER G., SHIRLEY P., HUBBARD P. M., GREENBERG D. P.: The irradiance volume. *IEEE Computer Graphics and Applications 18*, 2 (March/April 1998), 32–43.

[HPB06] HASAN M., PELLACINI F., BALA K.: Direct-to-indirect transfer for cinematic relighting. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2006) 25*, 3 (Aug. 2006).

[HSA91] HANRAHAN P., SALZMAN D., AUPPERLE L.: A Rapid Hierarchical Radiosity Algorithm. *Computer Graphics (Proc. of SIGGRAPH '91) 25*, 4 (July 1991).

[KAMJ05] KRISTENSEN A. W., AKENINE-MÖLLER T., JENSEN H. W.: Precomputed local radiance transfer for real-time lighting design. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2005) 24*, 3 (Aug. 2005).

[Kel97] KELLER A.: Instant radiosity. In *SIGGRAPH '97* (1997), pp. 49–56.

[Leh04] LEHTINEN J.: *Foundations of Precomputed Radiance Transfer*. Master's thesis, Helsinki University of Technology, Sept. 2004.

[LMM05] LUM E. B., MA K.-L., MAX N.: Calculating hierarchical radiosity form factors using programmable graphics hardware. *Journal of Graphics Tools 10*, 4 (2005).

[LSSS04] LIU X., SLOAN P.-P. J., SHUM H.-Y., SNYDER J.: All-frequency precomputed radiance transfer for glossy objects. In *Rendering Techniques (Proc. of EG Symposium on Rendering)* (2004), pp. 337–344.

[NPG05] NIJASURE M., PATTANAIK S. N., GOEL V.: Real-time global illumination on GPUs. *Journal of Graphics Tools 10*, 2 (2005).

[NRH03] NG R., RAMAMOORTHI R., HANRAHAN P.: All-frequency shadows using non-linear wavelet lighting approximation. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2003) 22*, 3 (July 2003), 376–381.

[NRH04] NG R., RAMAMOORTHI R., HANRAHAN P.: Triple product wavelet integrals for all-frequency relighting. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2004) 23*, 3 (Aug. 2004), 477–487.

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P.: Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2002) 21*, 3 (July 2002), 703–712.

[PDC*03] PURCELL T. J., DONNER C., CAMMARANO M., JENSEN H. W., HANRAHAN P.: Photon mapping on programmable graphics hardware. In *Graphics Hardware 2003* (2003), pp. 41–50.

[SC97] SHIRLEY P., CHIU K.: A low distortion map between disk and square. *Journal of Graphic Tools 2*, 3 (1997), 45–52.

[SDS96] STOLLNITZ E. J., DEROSE T. D., SALESIN D. H.: *Wavelets for Computer Graphics*. Morgan Kaufmann, 1996.

[SHHS03] SLOAN P.-P., HALL J., HART J., SNYDER J.: Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2003) 22*, 3 (2003).

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2002) 21*, 3 (July 2002).

[SLSS03] SLOAN P.-P., LIU X., SHUM H.-Y., SNYDER J.: Bi-scale radiance transfer. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2003) 22*, 3 (2003).

[SWS05] SVEN WOOP J. S., SLUSALLEK P.: RPU: A programmable ray processing unit for realtime ray tracing. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2005) 24*, 3 (Aug. 2005).

[WSB01] WALD I., SLUSALLEK P., BENTHIN C.: Interactive distributed ray tracing of highly complex models. In *Rendering Techniques 2001 (Proc. EG Workshop on Rendering)* (2001).

[WTL06] WANG R., TRAN J., LUEBKE D.: All-frequency relighting of glossy objects. *ACM Transactions on Graphics (to appear)* (2006).